

MANUEL UTILISATEUR GUS CLIENT 1.03

Auteur: Augustin Delale

Version: bêta

Date: 25/09/2014

Table des matières

Introduction.....	2
I. Utilisation basique.....	3
1. Récupérer le JAR de Gus Client.....	3
2. Premier lancement de Gus Client.....	3
3. Le code source de Gus Client.....	4
4. Extraire le code source.....	6
5. Compiler le code source.....	7
6. Déployer les projets test.....	9
7. Introduction au paramétrage.....	11
8. Introduction à la programmation gus06.....	13
9. Règles de programmation pour les entités.....	15
10. Votre première entité.....	16
11. Votre premier projet.....	18
12. Concernant la gestion des développements.....	20
13. Ajouter des caractéristiques aux entités.....	21
14. Gérer les erreurs de compilation.....	23
15. Faire appel à des services.....	25
16. Spécifier des règles de mapping.....	29
17. Mise au point avant de poursuivre.....	32
II. Utilisation avancée.....	33

Avertissement : Ce document est actuellement en cours de rédaction. La version bêta est mise en ligne à titre indicatif dans l'attente de la version 1.0.



Framework gus06

<http://gus06.ephian.net>

Introduction

Le framework **gus06** est un framework de développement Java open source qui a la particularité d'être extrêmement léger. Il se compose de 15 fichiers java regroupés dans un package unique dont la taille totale sur le disque dur ne fait que 60 Ko. Il constitue néanmoins un ensemble de spécifications permettant de produire du code source simple et hautement réutilisable.

Le framework gus06 n'est pas en lui-même un outil opérationnel permettant aux développeurs de créer rapidement et aisément des applications, pour la simple raison qu'il n'implémente aucune fonctionnalité. En revanche, le code source basé sur les spécifications gus06 permet d'obtenir de tels outils. Une première étape consiste par conséquent à créer un IDE basé sur le framework pour rendre ce dernier facilement utilisable.

Une première tentative est actuellement en cours: il s'agit du logiciel **Gus Client**. Si vous êtes curieux de savoir ce qu'il en retourne, je vous invite à télécharger ce logiciel sur le blog <http://gus06.ephian.net> et à poursuivre la lecture de ce document qui en constitue le manuel utilisateur.

Ce document s'adresse donc à tous les développeurs souhaitant découvrir et éprouver le framework gus06 en utilisant le logiciel Gus Client. Il adopte une approche très concrète et vient en complément d'un autre document plus théorique qui explique les différents concepts de la programmation gus06 : "Introduction à la programmation gus06".

Ce manuel s'organise en deux grandes parties :

1. Une première partie intitulée "Utilisation basique" qui présente toutes les fonctionnalités "clé en main" du logiciel Gus Client, permettant de développer et gérer son propre code source pour concevoir des applications.
2. Une deuxième partie intitulée "Utilisation avancée" qui s'intéresse aux possibilités de personnalisation et d'évolution du logiciel, nécessitant une compréhension plus poussée des mécanismes mis en jeu.

Il va sans dire que la deuxième partie est réservée aux utilisateurs de Gus Client ayant déjà une petite expérience du logiciel et animés par la curiosité de l'apprenti mécanicien qui regarde sous le capot de la voiture pour comprendre comment ça marche.

Il ne me reste plus qu'à vous souhaiter une bonne lecture !

Augustin



I. Utilisation basique

Vous seriez plutôt quel genre ?

- 1- "Pas de blabla ! je manipule et je comprend au fur et à mesure que j'avance"
- 2- "De temps en temps, quelques explications ne font pas de mal pour aider à la compréhension"

Un peu des deux ? Ca tombe bien ! Cette première partie adopte un mixte de ces deux genres en alternant les recettes de cuisines (1/ 2/ 3/ ...) et des passages explicatifs qui vous apporteront les bases nécessaires pour comprendre ce qu'il se passe. C'est parti !

1. Récupérer le JAR de Gus Client

Pour commencer, je vous invite à télécharger la version 1.03 du logiciel Gus Client en allant sur le blog <http://gus06.ephian.net> (lien disponible dans l'article daté du 21 septembre 2014). Le fichier correspondant est le suivant : gus06_1.03.jar (2.15 Mo)

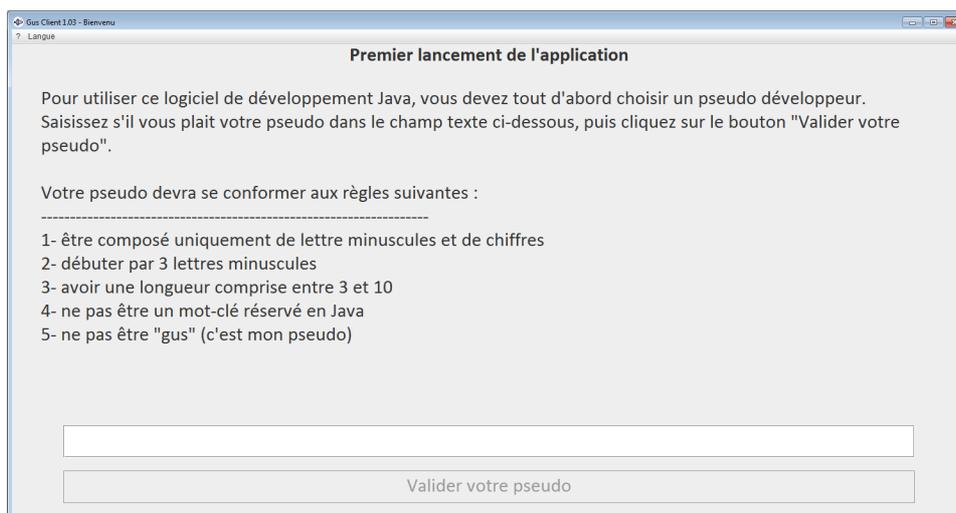
Assurez-vous ensuite que vous disposez bien sur votre machine d'une JRE 8 et que vous êtes capable d'exécuter le jar en double-cliquant dessus, ou par la ligne de commande suivante :

```
java -jar gus06_1.03.jar
```

2. Premier lancement de Gus Client

Au premier lancement, vous devriez voir apparaître ceci :

Dans le cas contraire... hum, lisez l'annexe 1 de ce manuel qui explique comment faire en cas d'erreur inhabituelle.



Pour utiliser le logiciel Gus Client, vous devez vous choisir un pseudo développeur qui identifiera votre code source sans risque de confusion avec le code source des autres. Lisez attentivement les instructions, puis tapez le pseudo que vous vous êtes choisi, et enfin validez en cliquant sur le bouton.

Le logiciel vous souhaite ensuite la bienvenue et vous fournit quelques explications pour vous permettre de commencer à l'utiliser. Plutôt qu'une bête capture d'écran, le mieux est encore de vous redonner le texte exacte :

Ce logiciel va vous permettre de développer rapidement des applications Java grâce au framework gus06. Il est organisé en différents espaces graphiques accessibles par le menu "Espaces". Vous vous trouvez actuellement dans l'espace "Bienvenu" qui se contente de présenter chacun des autres espaces.

Espace "Documentation" : Cet espace contient toute la documentation nécessaire pour utiliser efficacement ce logiciel et pour apprendre à programmer avec le framework gus06. Il comporte un guide utilisateur, une série de tutoriels, ainsi qu'un explorateur donnant accès à l'intégralité du code source de cette application.

Espace "Supervision" : Cet espace vous permettra de superviser l'état exacte de cette application, d'une part les informations liées à son exécution (notamment les données et les objets stockés en mémoire, ou encore l'état de la machine virtuelle), et d'autre part le contenu exacte du fichier JAR (paramétrage, code source, ressources, fichiers class). Ces informations vous serviront à comprendre le fonctionnement de l'application et à débiter vos futurs projets applicatifs.

Espace "Répertoires" : Cet espace vous permettra d'explorer de diverses manières le répertoire de votre ordinateur qui est utilisé par l'application pour enregistrer des données : **C:\GUS\GUS06\root**

Espace "Entités" : Cet espace vous permettra de gérer les composants de programmation Java liés au framework gus06 (appelés entités) que vous aurez développé ou récupéré à partir d'autres développeurs. Il permet notamment de visualiser, éditer, supprimer, renommer ou dupliquer des entités, mais aussi de les tester.

Espace "Projets" : Cet espace vous permettra de gérer vos projets de développement, qui peuvent être soit des applications, soit des bibliothèques de composants, soit des systèmes fonctionnels. Il permet notamment de compiler et déployer vos applications pour les rendre autonomes.

Notons que le chemin de répertoire surligné en jaune s'adapte à votre contexte d'utilisation.

Notons surtout que les fonctionnalités décrites ne correspondent pas à l'état d'avancement réel de Gus Client 1.03, mais à l'objectif final, et qu'il faut donc s'attendre à un certain décalage.

3. Le code source de Gus Client

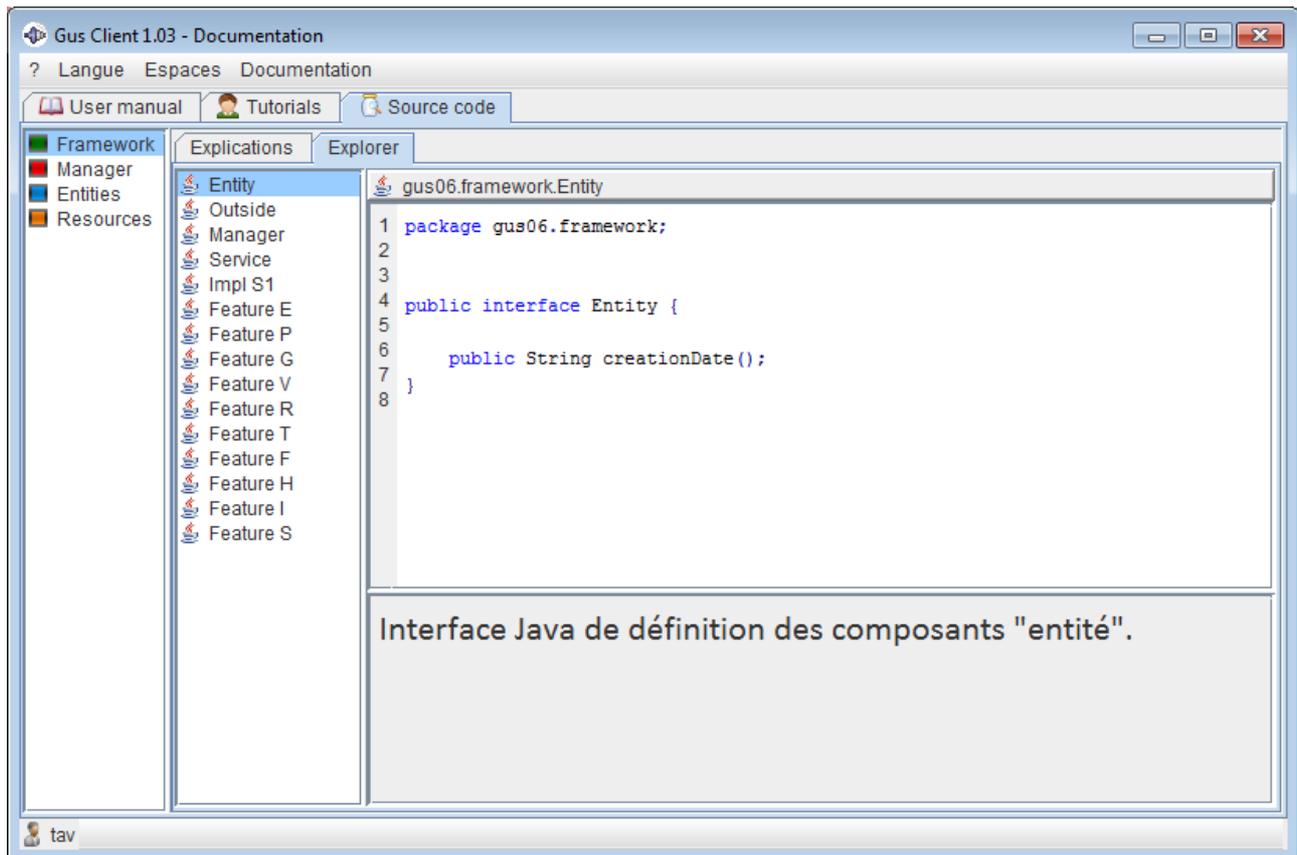
Rendez-vous sur l'espace "Documentation", puis cliquez sur l'onglet "Code source". Vous avez accès ici à l'ensemble du code source qui constitue l'application. Celui-ci se divise en 4 parties qui sont : **Framework, Manager, Entities, Resources**. Chacune d'elles est associée à une zone d'explications et un explorateur de contenu.



Intéressons-nous au code source du framework gus06. Comme dit précédemment, il se compose de 15 classes regroupées dans un package unique : gus06.framework

On y trouve 13 interfaces : Entity, Manager, Service, E, P, G, V, R, T, F, H, I, S
Ainsi que 2 implémentations : Outside, S1

Je vous invite à parcourir l'intégralité de ce code source. Cela ne devrait pas vous prendre plus d'une minute, et vous saurez désormais précisément de quoi on parle.



Je ne devrais pas vous poser cette question là mais...

A présent que vous avez fait connaissance avec le framework gus06, qu'en pensez-vous ? Quel intérêt y voyez-vous ? Quel code source dérivé produiriez-vous pour développer un outil semblable à Gus Client ?

Si vous êtes curieux de savoir comment j'ai répondu à la question, vous pouvez parcourir le reste du code source (mais ça risque de vous prendre un peu de temps parce que Gus Client 1.03 comporte en tout 1126 classes).

Soyons honnête: avoir pris connaissance des 15 classes du framework gus06 ne vous avance pas

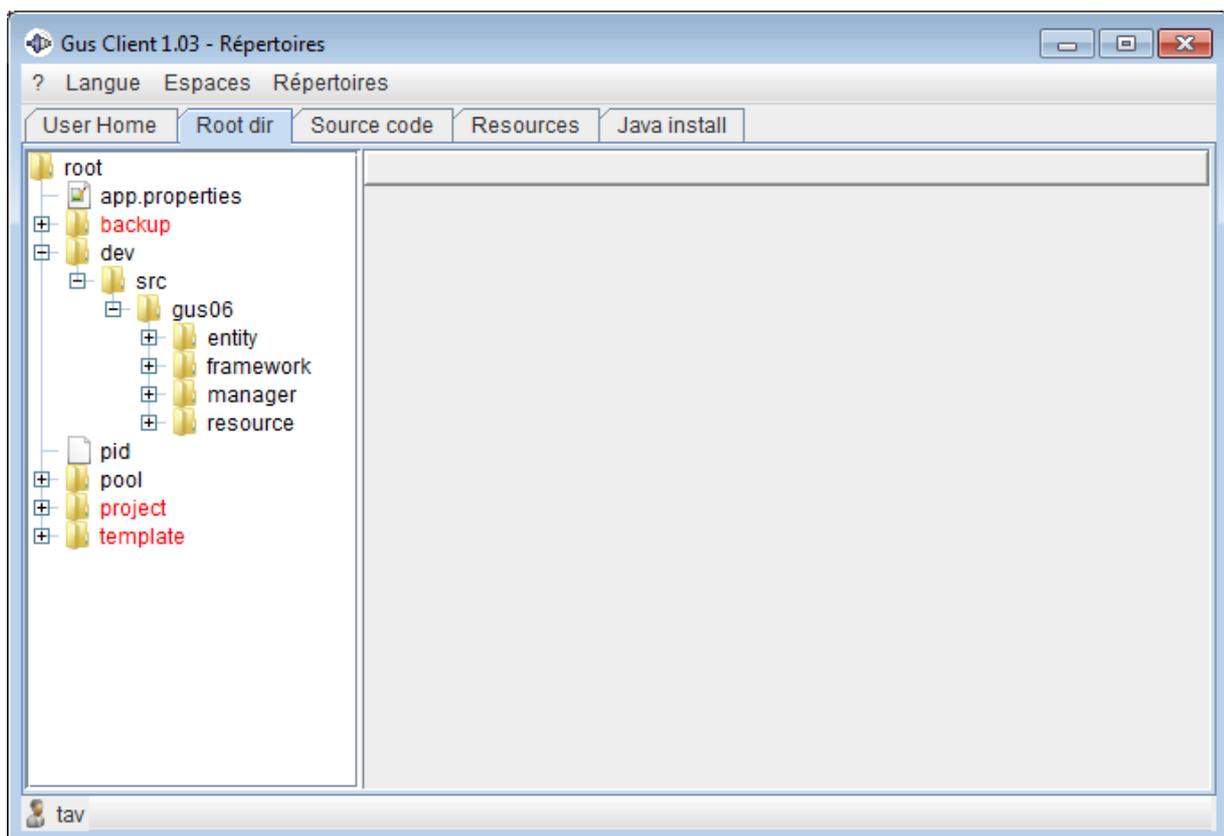


beaucoup pour appréhender la manière dont on peut concrètement en tirer profit. Et parcourir le reste du code source ne fera qu'ajouter à votre perplexité. A vrai dire, ce n'est pas en vous présentant le code source du framework gus06 que je pourrai espérer vous convaincre de son utilité mais en vous expliquant comment je m'en suis servi.

4. Extraire le code source

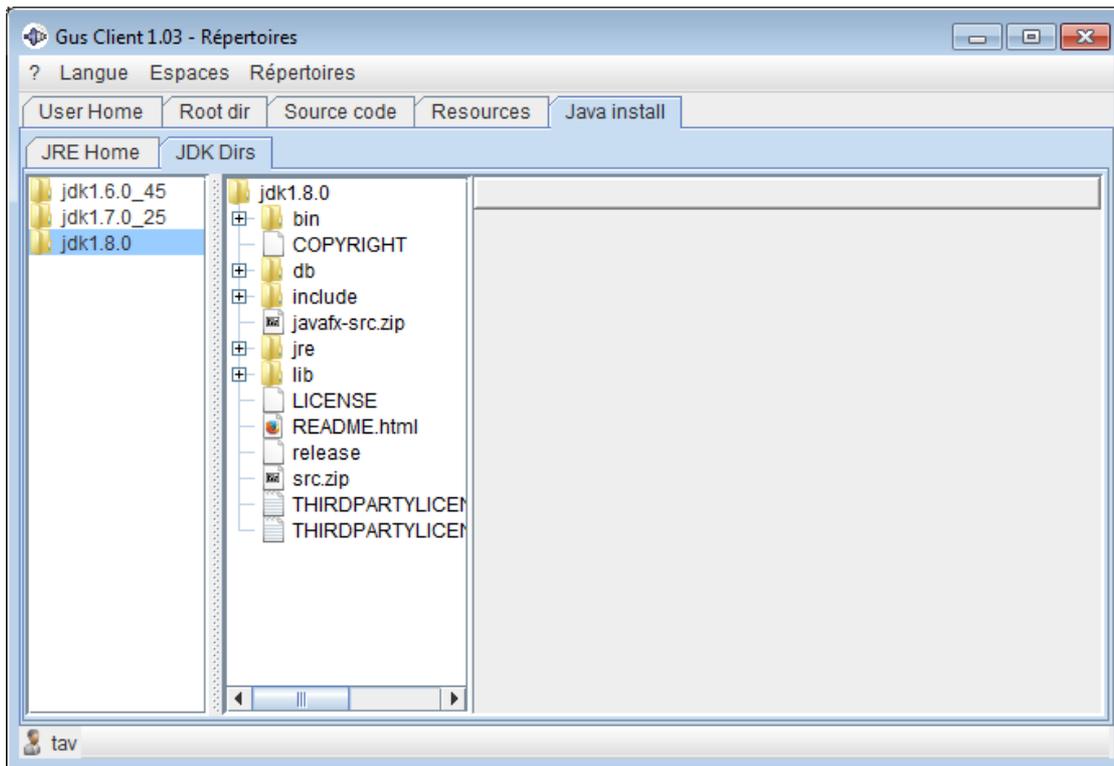
Le menu associé à l'espace Documentation permet d'extraire du JAR chacune des 4 parties de l'application. Vous pouvez cliquer successivement sur chaque item (les actions sont quasi-instantanées), puis vérifier que le code source et les fichiers de ressource ont bien été copiés sur votre disque dur, à la racine de l'emplacement suivant : <répertoire du JAR>/root/dev/src/

Ce code extrait est aussi accessible depuis le logiciel en vous rendant à l'onglet "Root dir" de l'espace "Répertoires". Un rafraîchissement de l'explorateur de fichiers (F5) sera sans doute nécessaire pour faire apparaître les répertoires nouvellement créés.



5. Compiler le code source

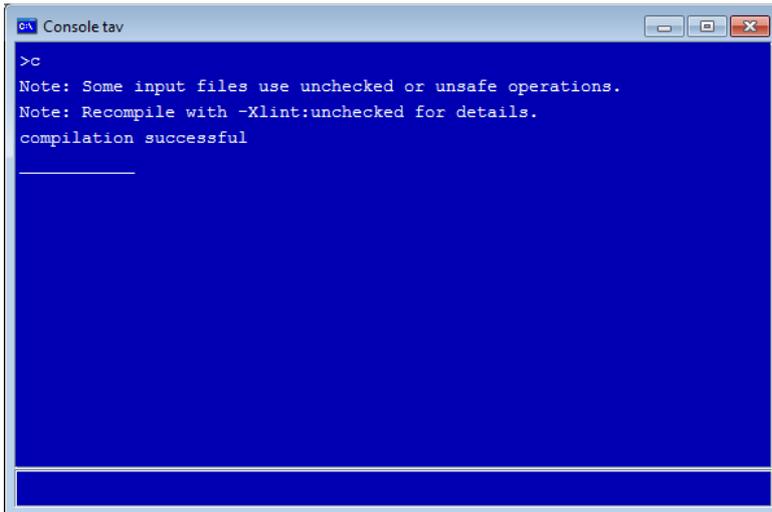
L'étape suivante consiste à compiler le code source que l'on vient d'extraire. Pour cela, un JDK (Java Development Toolkit) pour Java 6 ou supérieur doit être installé sur votre ordinateur. En vous rendant à l'onglet "Java Install / JDK Dirs" de l'espace "Répertoires" vous pourrez vérifier les différentes installations qui ont été trouvées par le logiciel.



Dans la version 1.03, aucun élément graphique (menu ou bouton) n'étant encore prévu pour effectuer la tâche de compilation générale, nous allons avoir recours à un outils habituellement réservé aux utilisateurs avancés : la console.

Dans le menu "?", cliquez sur "Affichez la console". Ensuite tapez la commande "c" suivi de ENTER. La compilation prend un certain temps (de l'ordre de 30s). Une fois terminée, vous devriez obtenir ceci :

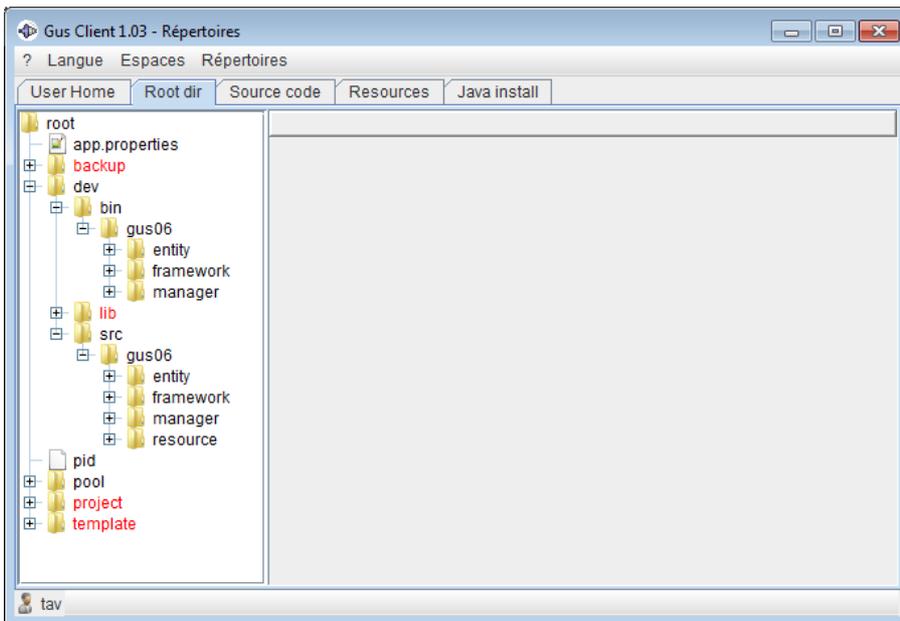




```
Console tav
>c
Note: Some input files use unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.
compilation successful
```

Dans le cas contraire... hum, lisez l'annexe 2

Après cela, si vous rafraichissez l'explorateur de l'onglet "Root dir" (F5) ...



Vous constaterez qu'un répertoire bin a été créé juste à côté du répertoire src, avec une arborescence de fichiers .class correspondant à l'arborescence de fichiers .java du côté src. Le sous-répertoire "resource" a quant à lui été oublié puisqu'il ne contient pas de fichiers java.

Votre répertoire dev contient désormais 2619 fichiers répartis comme suit :

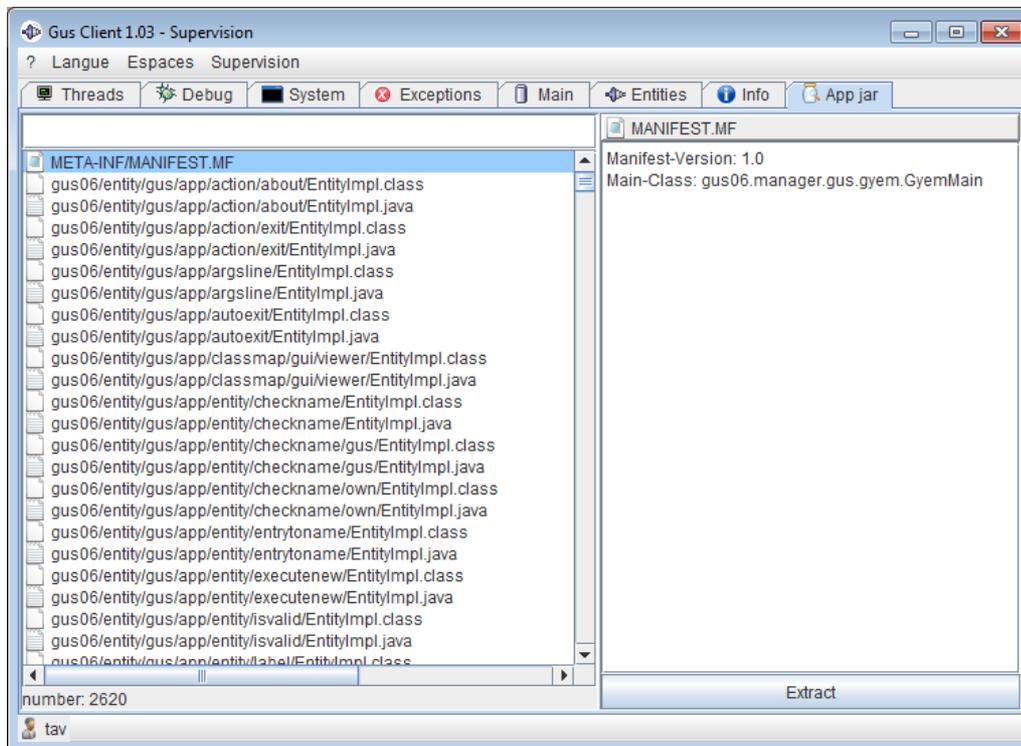
- 1342 fichiers .class
- 1126 fichiers .java
- 62 fichiers .gif
- 89 fichiers sans extension



Framework gus06
<http://gus06.ephian.net>

Etonnamment, on retrouve ici le contenu exacte du JAR gus06_1.03.jar à une exception près : il manque le fichier de manifest MANIFEST.MF

Pour information, vous pouvez accéder directement à l'intégralité des 2620 fichiers contenus dans le JAR à partir de l'onglet "App jar" de l'espace "Supervision".



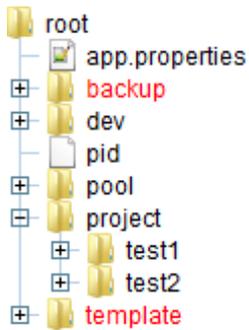
L'extraction et la compilation du code source constituaient des étapes préliminaires pour pouvoir démarrer des projets applicatifs. Nous allons commencer avec des projets test.

6. Déployer les projets test

Rendez-vous sur l'espace "Projets", puis, dans le menu associé, cliquez sur "Créer un nouveau projet" et choisissez comme nom de projet : "test1". Recommencez ensuite avec "test2". Vous pourrez créer de cette manière autant de projets que vous le désirez.

Pour chaque projet, le logiciel crée un répertoire spécifique dans le sous-répertoire "project" de votre répertoire root. Ce qui donne actuellement ceci :

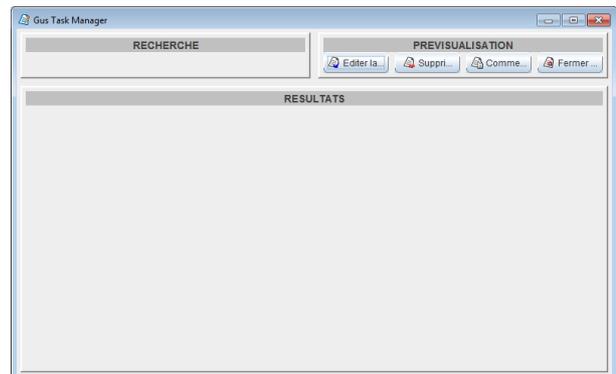
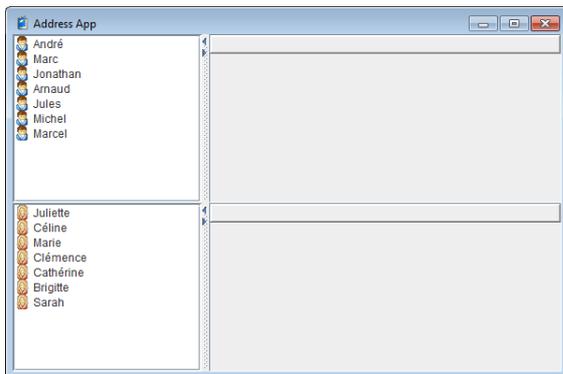




Un projet qui vient d'être créé est directement sélectionné et affiché dans l'espace "Projets", mais vous pouvez ensuite passer d'un projet à un autre grâce à l'item de menu "Sélectionner un projet".

Revenez au projet "test1" puis cliquez sur l'item "Importer le projet". Le nom n'est sans doute pas très pertinent puisqu'il s'agit en fait d'importer des ressources dans le projet courant. Gus Client 1.03 vous donne le choix entre deux noms d'application : **address** et **taskmanager**.

Choisissez par exemple d'importer "address" dans "test1" puis "taskmanager" dans "test2". Cliquez ensuite sur l'item "Déployer et tester le projet" pour chacun des projets. Et voilà le résultat dans chacun des cas !



Il faut bien avouer que ces applications sont loin d'être achevées mais c'était surtout histoire de lancer quelque chose ...



7. Introduction au paramétrage

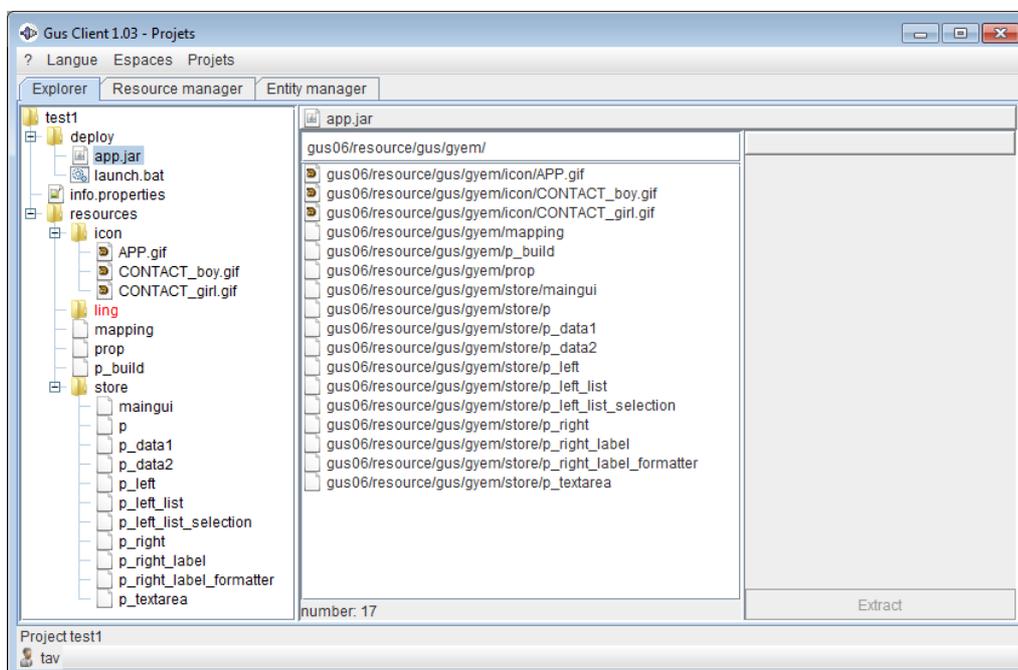
Nous avons pu vérifier que les deux projets fonctionnent correctement. Nous allons nous pencher à présent sur les fichiers contenus dans le projet test1 pour commencer à comprendre le paramétrage d'une application.

Sélectionnez le projet test1 (ou rafraichissez l'explorateur de fichiers). Il contient 2 sous-répertoires et un fichier properties.

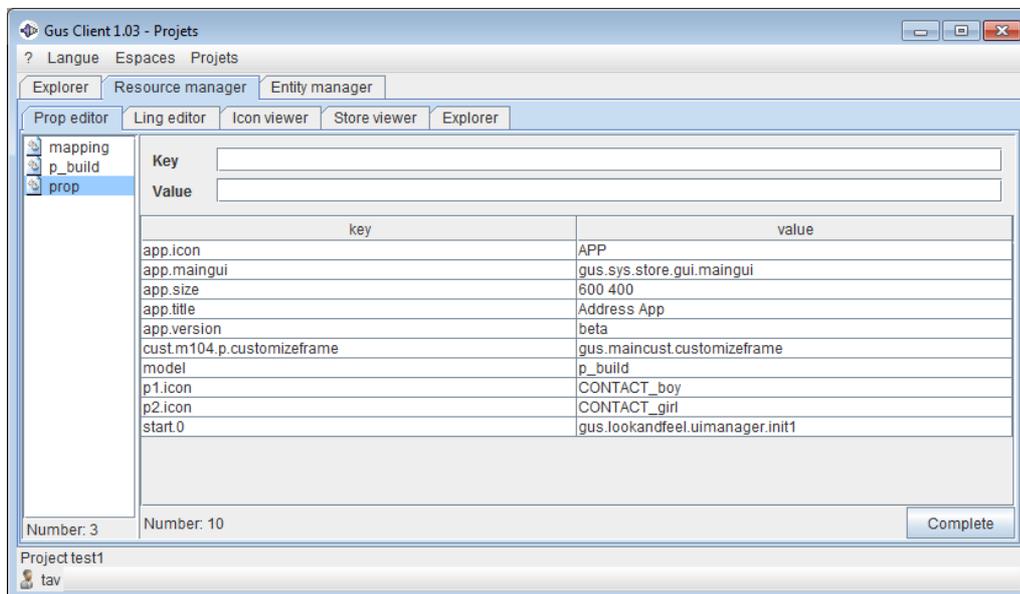
1. Le fichier **info.properties** n'a guère d'utilité si ce n'est de renseigner sur la date de création du projet.
2. Le sous-répertoire **deploy** (ajouté au moment du premier déploiement) contient le jar applicatif automatiquement généré.
3. Le sous-répertoire **resources** contient l'ensemble des fichiers de ressource nécessaire à l'application

Vous pouvez sélectionner le jar si vous êtes curieux de savoir ce qu'il contient. En tapant dans le champ de filtrage le chemin : "gus06/resource/gus/gyem", vous visualiserez l'ensemble des ressources contenues dans le jar.

En fait, l'application s'attend à trouver son paramétrage et ses ressources à partir de ce chemin racine, et son arborescence de fichiers coïncide avec celle du répertoire resources.



Le premier fichier de paramétrage auquel s'intéresse l'application est le fichier **prop**. Il s'agit d'un fichier qui contient les propriétés générales de l'application. Pour l'éditer facilement, choisissez plutôt l'onglet "Resource manager" plutôt que l'explorateur de fichiers.

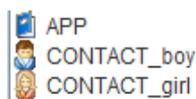


Si vous avez des questions relatives à l'utilisation de l'éditeur de propriétés, n'hésitez pas à consulter l'annexe ...

Voici les différentes propriétés permettant de modifier les éléments graphiques de la fenêtre principale de l'application.

Propriété	Explication
app.title	Titre de la fenêtre
app.size	Taille de la fenêtre (format:<largeur> <hauteur>)
app.icon	Identifiant de l'icône de la fenêtre
app.version	Numéro de version de l'application
app.maingui	Le nom de l'entité principale

L'identifiant d'icône correspond au nom du fichier gif cible sans l'extension, situé dans le répertoire icon. Le projet test1 contient 3 icônes visibles dans l'onglet "Resource manager / Icon viewer".



Pour tester le résultat de vos modifications, vous devrez à chaque fois redéployer et relancer l'application. Cette opération peut être exécutée depuis le menu, mais aussi grâce au raccourci clavier F9. Je vous conseille de le mémoriser car il vous sera très utile.

Petit exemple de modifications :

app.title = Mes contacts

app.icon = CONTACT_boy



Et la propriété `app.maingui` ? Il s'agit du nom de l'entité chargée de générer l'interface graphique principale de l'application. Il serait peut être temps de commencer à parler de la programmation gus06 et notamment de ces petites briques fonctionnelles qu'on appelle des entités.

8. Introduction à la programmation gus06

Vous avez pu constater que le code source du logiciel Gus Client se divisait en 4 parties. A vrai dire, cette structure se retrouvera invariablement dans toute application basée sur le framework gus06. Attardons-nous sur chacune de ces parties :

PARTIE 1 : Framework : On trouve ici les 15 classes java qui composent le framework gus06. D'une part, celui-ci définit deux type de composants qui sont l'entité et le gestionnaire. D'autre part, il offre un mécanisme générique permettant à ces composants d'interagir.

PARTIE 2 : Manager : On trouve ici les classes qui composent un unique gestionnaire dont le rôle est d'assurer les fonctions vitales de l'application concernant notamment la gestion des entités. Ce composant fait office de noyau de l'application. Pour le moment, je n'ai développé qu'un seul gestionnaire nommé Gyem, qui est systématiquement proposé pour chaque application.

PARTIE 3 : Entity : On trouve ici les classes des nombreuses entités utilisées par l'application. Ces petites briques de complexité très variable prennent en charge tous les aspects fonctionnels de l'application, qu'ils soient transverses ou métier.

PARTIE 4 : Resource : On trouve ici les fichiers de paramétrage utilisés par le gestionnaire ainsi que les autres ressources de l'application (images, icônes, sons, traductions...)

Pour un gestionnaire donné (en l'occurrence Gyem) chaque application tirera sa spécificité des parties 3 et 4 uniquement. C'est donc sur celles-ci que vous devrez travailler.



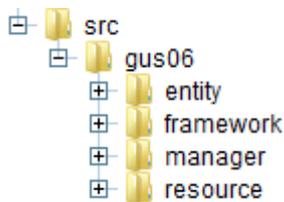
Framework gus06

<http://gus06.ephian.net>

Comme vous avez déjà pu le remarquer, les packages de chaque partie adoptent les conventions de nommage suivantes :

Partie	Package commençant par
Framework	gus06.framework
Manager	gus06.manager
Entity	gus06.entity
Resource	gus06.resource

Pour n'importe quelle application gus06, le répertoire de stockage des sources Java (à partir duquel les packages Java sont construits) ressemblera donc à cela :



Et ensuite ? Comment s'organisent les packages ?

Pour le framework, la question ne se pose pas puisqu'il n'y a que le package gus06.framework. Dans le cas des composants gus06 (gestionnaires et entités), le nom de package se poursuit systématiquement par le **pseudo** du développeur auteur du composant. Et pour les ressources, le nom du package est fixé par le gestionnaire qui gère l'application.

Mon pseudo étant gus :

- Mes entités commenceront toutes par : *gus06.entity.gus*.
- Mes gestionnaires commenceront tous par : *gus06.manager.gus*.

Imaginons maintenant que votre pseudo soit tav :

- Vos entités commenceront toutes par : *gus06.entity.tav*.
- Vos gestionnaires commenceront tous par : *gus06.manager.tav*.

Ceci étant, je vous conseillerai de vous focaliser sur le développement d'entités plutôt que de vous lancer dans le développement d'un ou plusieurs gestionnaires.

A priori, vos tâches en tant que développeur gus06 consisteront à créer de nouvelles entités et les paramétrer au sein de projets applicatifs, mais aussi à réutiliser des entités existantes.



9. Règles de programmation pour les entités

Voici les différentes règles auxquelles doit se conformer le code source d'une entité :

1. Le code source d'une entité est regroupé au sein d'un package unique.
2. Le nom d'une entité se déduit de son package de la manière suivante :
nom-package = gus06.entity.<nom-entité>
3. Le nom d'une entité commence par le pseudo du développeur qui en est l'auteur.
nom-entité = <pseudo>.<le reste est choisi par le développeur>
4. Le code source d'une entité doit comporter une classe EntityImpl appelée classe principale de l'entité (il se limite le plus souvent à cette seule classe).
5. La classe principale d'une entité doit implémenter l'interface Entity du framework.
6. La classe principale d'une entité ne peut contenir au plus qu'un seul constructeur sans paramètres.
7. L'implémentation de la méthode *creationDate()* doit renvoyer une chaîne de caractères correspondant à la date de création de l'entité au format *<aaaammjj>* (par exemple: 20141010).
8. Le code source d'une entité ne peut faire référence à du code appartenant à un autre composant gus06. Autrement dit, le seul import commençant par gus06 qui est autorisé dans les classes d'entité est : *import gus06.framework.*;*

Toutes ces règles peuvent paraître contraignantes mais en pratique elles sont toutes prises en compte par le mécanisme de génération automatique que vous utiliserez à chaque fois que vous créerez une nouvelle entité.

Ainsi, vous n'aurez plus qu'à vous soucier de la cohérence de nommage de vos entités (et accessoirement de leurs implémentations).

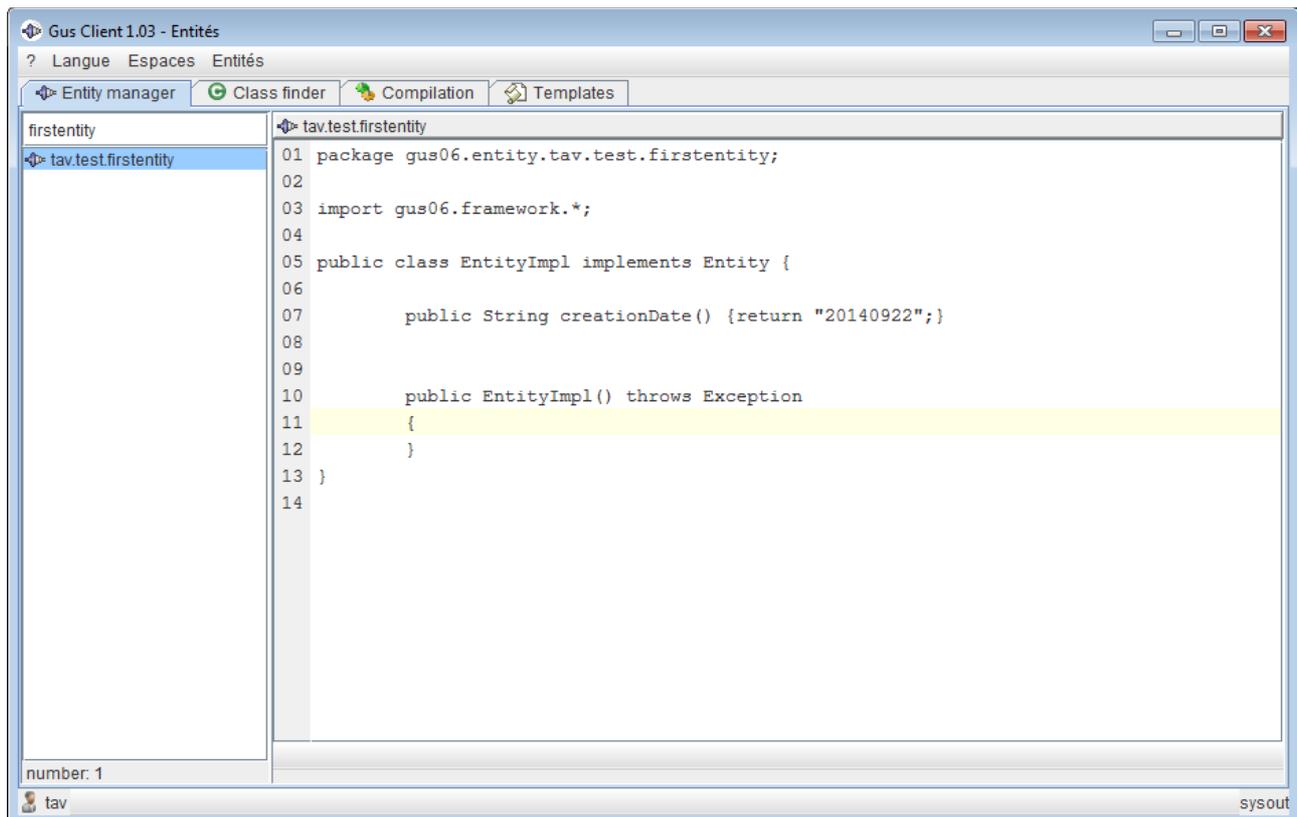
Bien, peut être serait-il temps de mettre tout cela en pratique !



10. Votre première entité

Je vous propose de développer votre première entité. Pour cela, rendez-vous dans l'espace "Entités" et cliquez sur l'item "Créer une nouvelle entité" du menu associé. Comme nom, prenez par exemple : "test.firstentity"

Que s'est-il passé ? Pour le savoir, recherchez votre nouvelle entité en tapant "firstentity" dans le champ texte qui filtre la liste des entités, et enfin sélectionnez votre entité. Vous devriez obtenir ceci (en remplaçant naturellement tav par votre propre pseudo)



The screenshot shows the 'Gus Client 1.03 - Entités' window. The 'Entity manager' tab is active, displaying a list of entities. The entity 'firstentity' is selected, and its details are shown in the right pane. The details pane shows the package 'gus06.entity.tav.test.firstentity' and the class 'EntityImpl' which implements the 'Entity' interface. The code for 'EntityImpl' is as follows:

```

01 package gus06.entity.tav.test.firstentity;
02
03 import gus06.framework.*;
04
05 public class EntityImpl implements Entity {
06
07     public String creationDate() {return "20140922";}
08
09
10     public EntityImpl() throws Exception
11     {
12     }
13 }
14

```

The status bar at the bottom shows 'number: 1' and the user 'tav'.

Le code source initial de votre entité vient d'être généré automatiquement et il est facile de constater que toutes les règles énoncées précédemment ont été respectées. Par ailleurs, le nom choisi (test.firstentity) n'étant pas valide, le générateur a de lui-même corrigé l'erreur en le faisant commencer par votre pseudo.

Il faudrait juste ajouter un peu de code histoire que votre entité fasse quelque chose. Débutons par un truc très simple : beep dans le constructeur !

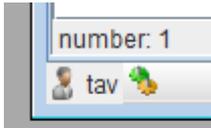
Dans le constructeur, ajoutez l'instruction suivante :
Toolkit.getDefaultToolkit().beep();



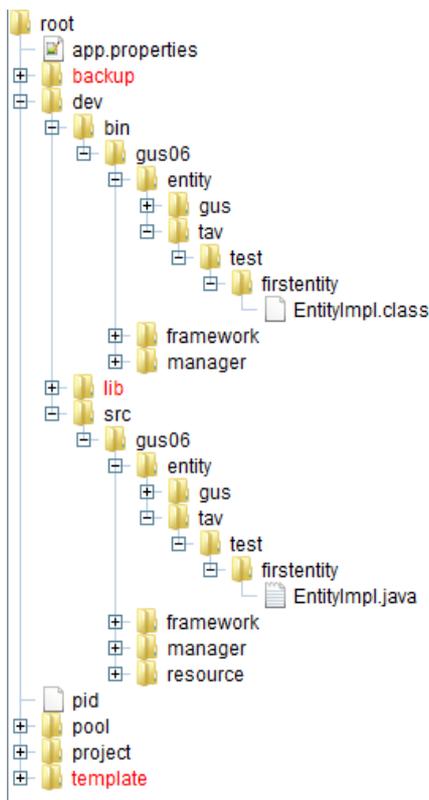
Il manque l'import pour la classe Toolkit ? Appuyez sur la touche F2 lorsque vous venez de taper "Toolkit" ou après avoir remis le curseur dessus, pour régler le problème.

Faites ensuite ctrl-m dans l'éditeur pour compiler votre entité.

Pendant la phase de compilation, vous remarquerez l'apparition de l'icône 🍌 en bas à gauche de la fenêtre.



Lorsque l'icône a disparu, vous pouvez aller vérifier dans l'explorateur du répertoire root que des fichiers *EntityImpl.java* et *EntityImpl.class* sont bien présents aux emplacements attendus, comme ci-dessous :



Il ne reste plus qu'à tester cette entité dans un projet applicatif.

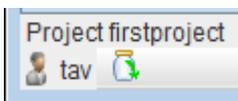


11. Votre premier projet

Après avoir développé votre première entité, il vous faut un projet pour la tester.

1. Revenez donc à l'espace "Projets", mais plutôt que d'utiliser le menu, appuyez cette fois sur la touche F12.
2. Créez un nouveau projet "firstproject".
3. Dans l'onglet "Resource manager / Prop editor", sélectionnez le fichier prop.
4. Ajoutez la propriété start = tav.test.firstentity
Ceci indique au gestionnaire qu'il doit instancier votre entité au démarrage de l'application.
5. Ajoutez aussi app.guidisabled = true
Ca, c'est pour indiquer que l'application n'est pas graphique.
6. Appuyez sur F9 pour tester.
Après quelques secondes, vous devriez entendre "Beep".

Pendant la phase de déploiement du JAR, vous remarquerez l'apparition de l'icône  en bas à gauche de la fenêtre.



A présent, effectuons quelques modifications dans notre code.

1. Revenez à l'espace "Entités" en appuyant sur la touche F11
2. Modifiez le code source de l'entité pour qu'il fasse 2 beeps (voir ci-dessous)
3. Faites ctrl-m dans l'éditeur, et attendez que l'icône  disparaisse
4. Appuyez sur F9, et attendez que l'icône  disparaisse
5. Vous devriez entendre "Beep.. Beep..."



Voici à quoi doit ressembler le code de votre entité avant que vous ne le compiliez.

```

package gus06.entity.tav.test.firstentity;

import gus06.framework.*;
import java.awt.Toolkit;

public class EntityImpl implements Entity {

    public String creationDate() {return "20140922";}

    public EntityImpl() throws Exception
    {
        Toolkit.getDefaultToolkit().beep();
        Thread.sleep(800);
        Toolkit.getDefaultToolkit().beep();
    }
}
    
```

Petite astuce : pour dupliquer la ligne de votre curseur, faites ctrl-e.

Et pour sauvegarder les modifications ? Inutile de faire un ctrl-s. L'éditeur sauvegarde automatiquement vos changements sans que vous n'ayez à vous en soucier. Et lorsque vous souhaitez revenir en arrière, vous disposez toujours du ctrl-z.

Voici un récapitulatif des raccourcis clavier propres à l'éditeur d'entités :

Raccourci clavier	Explication
Ctrl-z	Annule la dernière modification de texte
Ctrl-y	Remet la dernière modification de texte (annule le ctrl-z)
Ctrl-e	Duplique la ligne du curseur
Ctrl-d	Supprime la ligne du curseur
Ctrl-m	Compile l'entité
F2	Ajoute automatiquement l'import de la classe qui vient d'être tapée

Voici pour finir un récapitulatif des 3 raccourcis clavier qui sont valables sur tous les espaces :

Raccourci clavier	Explication
F12	Aller à l'espace suivant
F11	Aller à l'espace précédent
F9	Déployer et tester le projet courant



12. Concernant la gestion des développements

Avant d'aller plus loin, il est peut être utile de réfléchir un instant à la manière dont le logiciel Gus Client gère les développements Java en mettant en lumière les avantages et les inconvénients du système actuel.

Vous aurez remarqué que l'ensemble du code source Java est regroupé dans le répertoire `root\dev\src`, ce qui inclut les sources du framework, du gestionnaire, et de toutes les entités (tous projets et tous auteurs confondus). L'unicité des noms des composants qui entraîne l'unicité des packages Java permet effectivement de regrouper tout le code dans un répertoire unique sans risque de conflit.

Le soucis vient de la gestion des ressources liées aux différentes applications. Le répertoire `root\dev\src\gus06\resource` contient actuellement les fichiers ressource de Gus Client mais ne peut en aucune manière accueillir simultanément les ressources rattachées à plusieurs projets puisque d'un projet à l'autre, des fichiers de contenus distincts occupent a priori le même emplacement avec le même nom.

Bien que ça manque un peu de cohérence, le répertoire `src` sera vu comme un espace de stockage multi-projets à l'exception de la partie "ressources" qui reste spécifique à Gus Client. Les ressources spécifiques à chaque projet sont quant à elles stockées dans les emplacements dédiés : `root\project\<project-name>\resources\`

Lorsque Gus Client procède au déploiement du projet courant, il est configuré pour :

- rechercher les fichiers java dans `root\dev\src\`
- rechercher les fichiers class dans `root\dev\bin\`
- rechercher les fichiers de ressource dans `root\project\<project-name>\resources\`

Par ailleurs, dans le cas particulier des ressources, le mécanisme de déploiement fait coïncider la racine du répertoire de stockage avec le chemin racine "gus06/resource/gus/gyem" à partir duquel le gestionnaire Gyem recherche les ressources de l'application.

Ce système évoluera sans doute dans des versions futures de Gus Client mais il était important que vous compreniez ce qu'il en est actuellement.



13. Ajouter des caractéristiques aux entités

Pour le moment, notre entité ne peut pas être manipulée par le gestionnaire. Pour la rendre utilisable, nous devons lui ajouter une ou plusieurs caractéristiques, c'est à dire lui faire implémenter une ou plusieurs des 10 interfaces Java définies par le framework (E, P, G, V, R, T, F, H, I, S) avec éventuellement l'interface `java.lang.Runnable`.

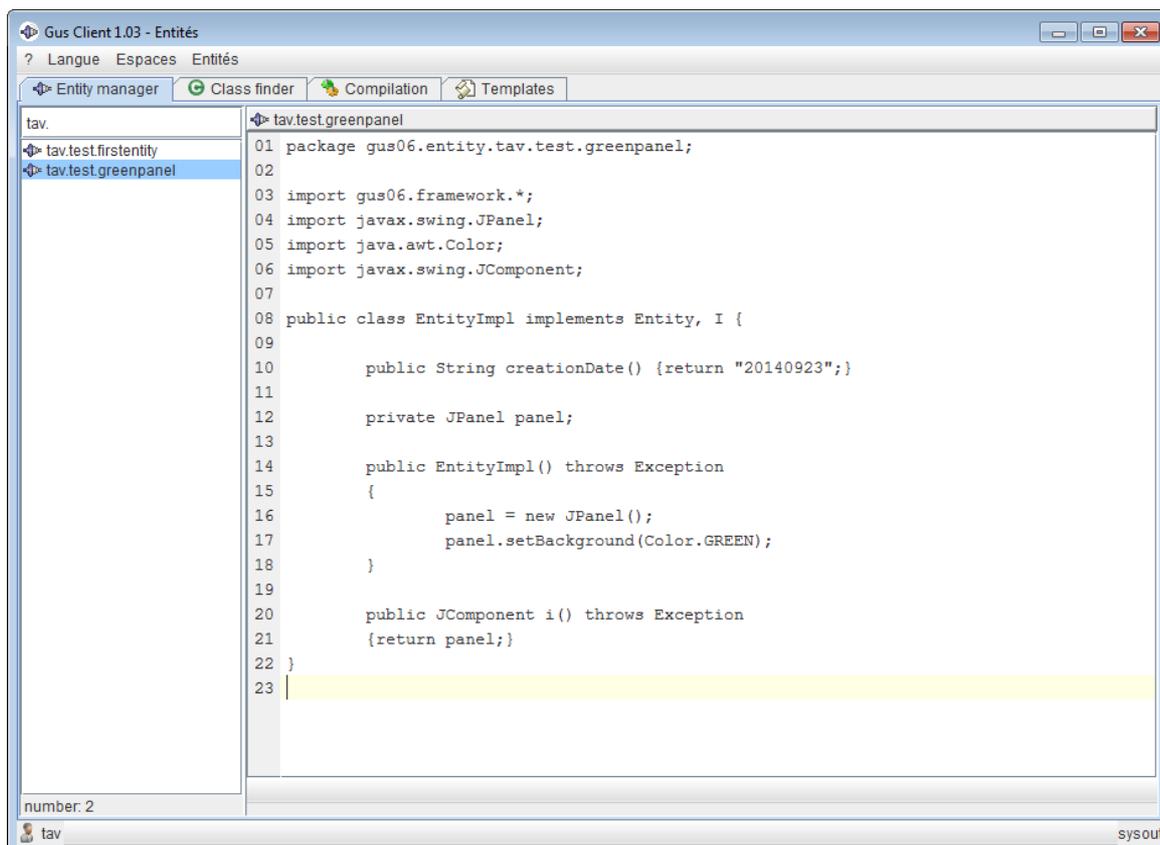
La première caractéristique que je vous propose d'utiliser est la caractéristique graphique correspondant à l'interface suivante :

```
package gus06.framework;

import javax.swing.JComponent;

public interface I {
    public JComponent i() throws Exception;
}
```

1. Pour cela, créez une nouvelle entité que vous appellerez "test.greenpanel" (en fait son nom sera <pseudo>.test.greenpanel), et complétez son code source pour obtenir le résultat suivant :



```
01 package gus06.entity.tav.test.greenpanel;
02
03 import gus06.framework.*;
04 import javax.swing.JPanel;
05 import java.awt.Color;
06 import javax.swing.JComponent;
07
08 public class EntityImpl implements Entity, I {
09
10     public String creationDate() {return "20140923";}
11
12     private JPanel panel;
13
14     public EntityImpl() throws Exception
15     {
16         panel = new JPanel();
17         panel.setBackground(Color.GREEN);
18     }
19
20     public JComponent i() throws Exception
21     {return panel;}
22 }
23
```



2. Pensez bien à appuyer sur F2 autant de fois que nécessaire pour vous épargner l'ajout manuel des imports. Puis terminez en compilant l'entité (ctrl-m).
3. Revenez à l'espace "Projets" (F12)
4. Editez le fichier prop en remplaçant la propriété app.guidisabled par :
app.maingui = tav.test.greenpanel
5. Testez votre projet (F9)
Vous entendez toujours beep beep, et vous avez en prime un joli panneau vert !



Détaillons ce qui vient de se passer :

1. Le gestionnaire Gyem détecte qu'il dispose d'une propriété start=tav.test.firstentity
2. Il instancie donc l'entité tav.test.firstentity, ce qui provoque le double-beep
3. Il remarque ensuite qu'il dispose d'une propriété app.maingui=tav.test.greepanel
4. En conséquence, il instancie l'entité tav.test.greenpanel, récupère l'objet JComponent renvoyé par la méthode i(), et place cet objet dans la fenêtre principale de l'application
5. Finalement, il rend la fenêtre visible.

Essayons ensuite de personnaliser cette fenêtre histoire de mettre en pratique les notions de paramétrage vues précédemment.

Pour le titre et la taille de la fenêtre, c'est très simple puisqu'il suffit d'ajouter dans le fichier prop les propriétés correspondantes. Par exemple :

```
app.title = Panneau vert  
app.size = 250 250
```

Pour l'icône en revanche, c'est plus compliqué parce que le gestionnaire Gyem ne dispose pas en natif d'un mécanisme de gestion d'icônes. Pour remédier à cette situation, il faut personnaliser le gestionnaire en ajoutant un paramétrage spécial (ce qui était déjà fait dans les applications test) :



Dans le fichier **prop**, ajoutez :

- `cust.m104.p.customizeframe = gus.maincust.customizeframe`

Dans le fichier **mapping**, ajoutez :

- `@inside = m010.t.inside`
- `@prop = call.g#m004.g.prop`

Pour le moment, vous devez juste savoir que ce paramétrage fait appel à des notions avancées qui seront étudiées dans la deuxième partie de ce manuel.

Bien! Il ne vous reste plus qu'à récupérer quelque part un fichier gif 16x16, le placer dans le répertoire "icon" dédié au projet : `root\project\firstproject\resources\icon\` et ajouter la propriété suivante : `app.icon = <nom-de-votre-fichier-gif-sans-extension>`

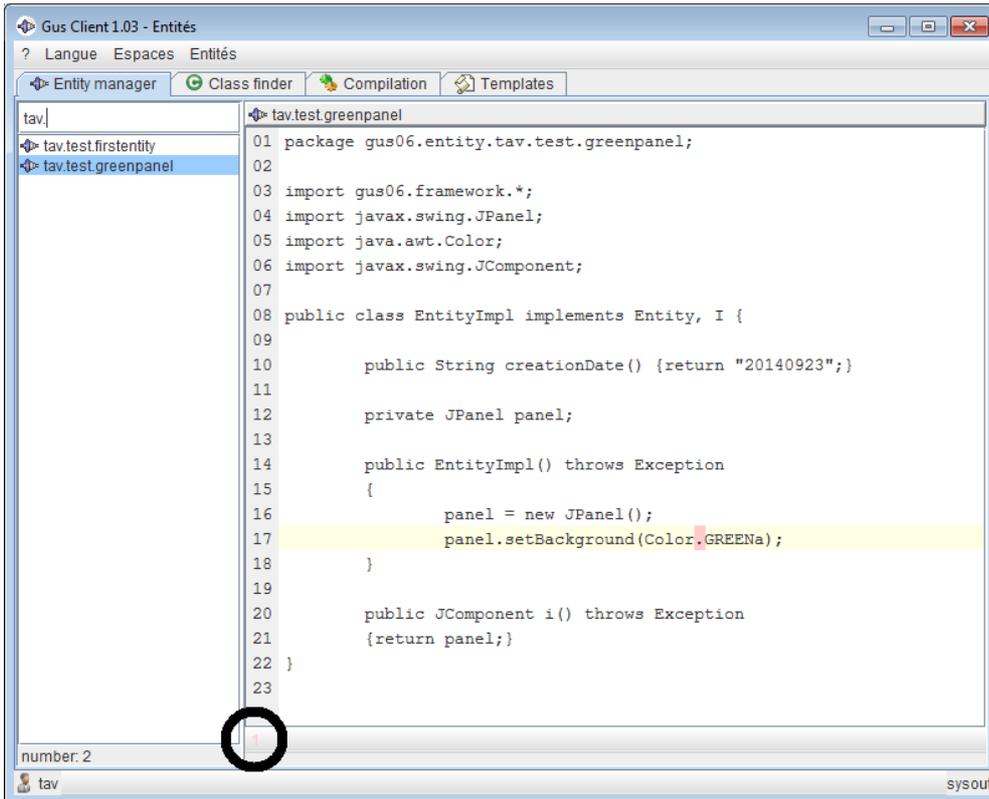
Pour obtenir le résultat ci-dessous, j'ai pris l'icône  :



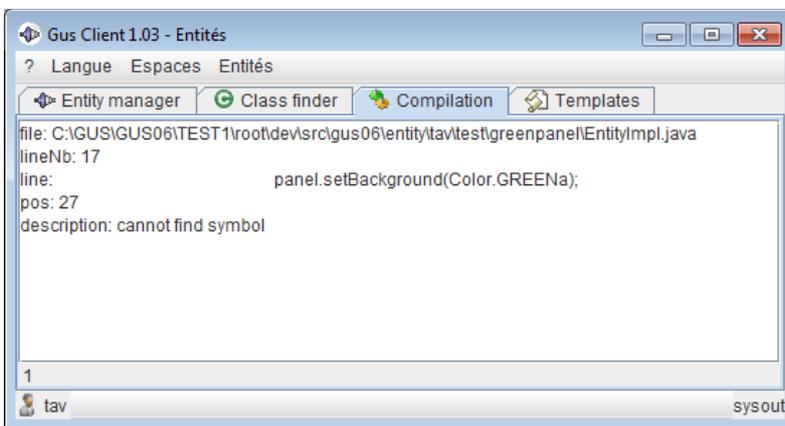
14. Gérer les erreurs de compilation

Lorsque vous compilez une entité, il arrive parfois que la compilation plante parce que vous avez mal tapé votre code, fait une erreur de syntaxe, ou simplement oublié un import. Dans ce cas, le logiciel Gus Client vous indique le ou les endroits de votre code qui posent problème par un surlignage rose. Le nombre de ces surlignages est par ailleurs mentionné dans la barre du bas (zone entourée en noir ci-dessous).





L'onglet "Compilation" vous donne quant à lui quelques détails sur le type d'erreur (en l'occurrence ici, symbole inconnu). Ce qui permet de mieux comprendre le problème.



Mais je ne vous cache pas que cet onglet (tout comme le processus général de compilation) devrait beaucoup évoluer dans des futures versions de Gus Client. Pour le moment, je vous montre tout cela faute de mieux.

Attention, quand une telle situation arrive, je vous conseille de corriger directement l'erreur et de relancer la compilation (ctrl-m) jusqu'à ce que vous soyez certain que tout s'est bien passé.



Pourquoi, me demanderez-vous ? Parce que le compilateur n'a pas de mémoire et qu'il oubliera que vous avez une entité dont la compilation a échoué alors que vous êtes peut être passé à une autre entité qui, elle, se compile correctement.

Mais il y a pire... Une entité qui échoue à la compilation perd le fichier EntityImpl.class qui lui était associé (qui ne sera recréé qu'une fois la compilation effectuée avec succès). Cette "désynchronisation" entre les fichiers java et les fichiers class n'étant pas détectée au moment du déploiement du projet courant, on se retrouve généralement avec un JAR incomplet qui a toutes les chances de planter à l'exécution.

Si vous vous retrouvez dans une telle situation, ou tout simplement que vous avez modifié certaines de vos entités en oubliant de les recompiler, je vous conseille vivement de tout remettre d'équerre par une compilation générale de tout le code source (c + enter dans la console). Ça prend 30 secondes mais ça permet de repartir sur de bonnes bases (ou à défaut de détecter les différentes entités buggées pour les corriger une à une).

15. Faire appel à des services

Les services sont l'étape suivante dans la découverte de la programmation gus06. De quoi s'agit-il ? Les services sont des objets Java qui implémentent l'interface Service du framework, dont je vous redonne ici le code source :

```
package gus06.framework;

public interface Service extends Runnable, E, F, G, H, I, P, R, S, T, V {}
```

Ces objets qui implémentent l'ensemble des caractéristiques définies par le framework servent de "passerelle" aux entités qui souhaitent déléger des traitements au gestionnaire, assurant un découplage entre le code appelant les méthodes et l'implémentation des méthodes.

En effet, les services agissent comme des enveloppes masquant l'objet qui implémente le ou les caractéristiques. L'implémentation peut être une entité quelconque ou bien pourquoi pas, un objet interne du gestionnaire, qui sait. L'entité qui obtient un service de la part du gestionnaire n'a aucune emprise sur l'origine de ce service, et cette séparation nette entre le code l'appelant et le code l'appelé est l'idée maitresse sur laquelle repose la flexibilité du code.

En pratique, l'entité fait appel à la méthode statique *service(Entity,String)* de la classe *Outside*. Mais un petit exemple sera certainement plus parlant qu'une longue explication. Pour expérimenter cette notion, je vous propose de faire un peu de composition graphique.



1. Remplacez-vous dans l'espace "Entités" et tapez "tav." dans le champ de filtrage (en substituant tav par votre pseudo).
2. Sélectionnez l'entité `tav.test.greenpanel` et appuyez sur F3 pour la dupliquer. Choisissez comme nouveau nom : `tav.test.bluepanel`.
3. Répétez cette opération 2 autres fois avec des noms similaires se terminant par `redpanel` et `blackpanel` (vous ne les voyez pas encore apparaître mais c'est normal).
4. Faites F5 pour rafraichir la liste des entités.
5. Pour chaque nouvelle entité, remplacez dans le code GREEN par la couleur adéquate (BLUE, RED, BLACK).
6. Pensez à compiler après chaque modification !
7. Créez ensuite une nouvelle entité à partir du menu en saisissant le texte suivant : `test.gridpanel22 i`
8. Rafraichissez encore une fois et sélectionnez cette dernière entité.

Voici ce que vous devez normalement obtenir :

The screenshot shows the 'Gus Client 1.03 - Entités' window. On the left, the 'Entity manager' tab is active, displaying a list of entities under the 'tav.' namespace: `tav.test.blackpanel`, `tav.test.bluepanel`, `tav.test.firstentity`, `tav.test.greenpanel`, `tav.test.gridpanel22` (highlighted), and `tav.test.redpanel`. On the right, the source code for `tav.test.gridpanel22` is displayed, showing a package declaration, imports for `gus06.framework.*` and `javax.swing.JComponent`, and a class `EntityImpl` implementing `Entity` and `I`. The class contains methods `creationDate()`, `EntityImpl()`, and `i()`.

```

01 package gus06.entity.tav.test.gridpanel22;
02
03 import gus06.framework.*;
04 import javax.swing.JComponent;
05
06 public class EntityImpl implements Entity, I {
07
08     public String creationDate() {return "20140923";}
09
10
11     public EntityImpl() throws Exception
12     {
13     }
14
15
16     public JComponent i() throws Exception
17     {
18         return null;
19     }
20 }
21

```

At the bottom left, the status bar shows 'number: 6' and 'tav'. At the bottom right, it shows 'sysout'.



Avant de compléter le code source de notre nouvelle entité, arrêtons nous un instant sur la liste filtrante des entités qui se trouve dans l'onglet "Entity Manager" de l'espace "Entités".

Voici un récapitulatif de raccourcis clavier propres à cette zone :

Raccourci clavier	Explication
F1	Créer une nouvelle entité
F2	Renommer l'entité sélectionnée
F3	Dupliquer l'entité sélectionnée
F5	Rafraichir la liste
DELETE	Supprimer l'entité sélectionnée

F5 est un raccourci indispensable pour faire apparaître les modifications dans la liste (nouvelle entité, renommage, suppression...). C'est dommage que se rafraichissement ne soit pas automatique mais ça viendra un jour, je vous le promet.

F2, F3 et DELETE sont très pratiques pour exécuter les opérations de maintenance classiques que sont le renommage, la duplication et la suppression d'entités.

F1 permet d'ajouter une nouvelle entité, et si je me suis abstenu jusqu'à présent de vous faire utiliser ce raccourci c'est qu'il existe un risque non négligeable de se mélanger avec un autre raccourci F1 actif lui à partir de l'éditeur d'entité (qui déclenche une action complètement différente que nous n'avons pas encore vu). Vous pouvez naturellement utiliser F1 pour créer vos entités mais assurez vous toujours avant que c'est bien la liste qui a le focus de votre clavier et non l'éditeur.

A propos de la création d'entité, vous aurez remarqué qu'il est possible de faire suivre le nom par une ou plusieurs lettres de caractéristiques pour générer une classe implémentant ces caractéristiques. Ca fait gagner du temps quand on sait à l'avance quel type d'entité on souhaite créer.

Revenons à notre entité `tav.test.gridpanel22`. Je vous propose de compléter son code source comme suit : (pensez à utiliser F2 à chaque fois que nécessaire)

```
package gus06.entity.tav.test.gridpanel22;
```

```
import gus06.framework.*;
import javax.swing.JComponent;
import javax.swing.JPanel;
import java.awt.GridLayout;
```



```

public class EntityImpl implements Entity, I {

    public String creationDate() {return "20140923";}

    private Service green;
    private Service red;
    private Service blue;
    private Service black;

    private JPanel panel;

    public EntityImpl() throws Exception
    {
        green = Outside.service(this,"tav.test.greenpanel");
        red = Outside.service(this,"tav.test.redpanel");
        blue = Outside.service(this,"tav.test.bluepanel");
        black = Outside.service(this,"tav.test.blackpanel");

        panel = new JPanel(new GridLayout(2,2));

        panel.add(green.i());
        panel.add(red.i());
        panel.add(blue.i());
        panel.add(black.i());
    }

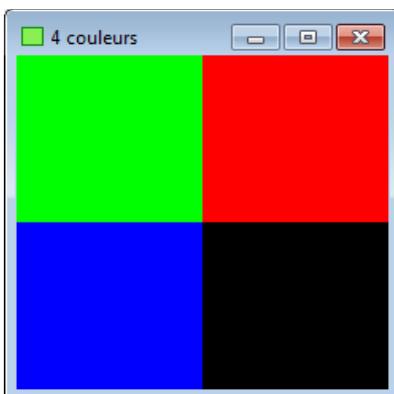
    public JComponent i() throws Exception
    {return panel;}
}

```

N'oubliez pas de compiler votre entité puis allez modifier le paramétrage de votre projet comme ceci :

- supprimez la propriété start
- app.maingui = tav.test.gridpanel22
- app.title = 4 couleurs

Lancez l'application en appuyant sur F9 et...



Passons aux explications.

La récupération d'un service se fait généralement dans le constructeur de l'entité, de la manière suivante :

```
var = Outside.service(this,"<identifiant-d-appel>");
```

ou var est une variable de classe de type Service et <identifiant-d-appel> une chaîne de caractères qui identifie le besoin de l'entité.

Il se trouve que les identifiants d'appel utilisés dans l'exemple ci-dessus correspondent à des noms d'entité et que c'est justement ces entités-là qui sont utilisées comme implémentations des services.

Cela est dû au mécanisme de génération de services propre au gestionnaire GYEM qui choisit par défaut d'interpréter l'identifiant comme un nom d'entité, mais ce n'est en aucun cas une règle générale. De fait, il serait trompeur de penser que la relation de dépendances est écrite en dur dans le code source de l'entité.

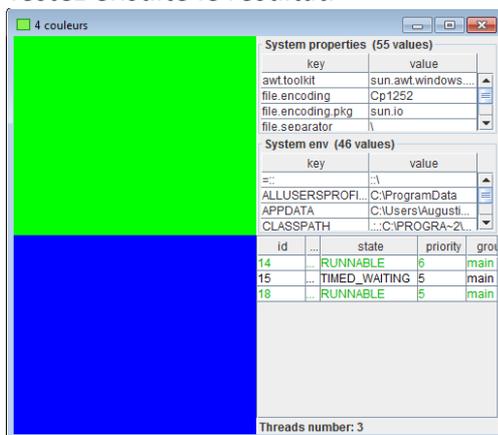
Il est en effet possible de paramétrer des règles de mapping entre les appels de service et les règles de génération de ces services, comme nous allons le voir juste après.

16. Spécifier des règles de mapping

Dans le fichier de mapping de votre projet, ajoutez les informations suivantes :

- tav.test.gridpanel22@tav.test.blackpanel = gus.thread.gui.viewer
- tav.test.gridpanel22@tav.test.redpanel = gus.system.prop.gui.viewer

Testez ensuite le résultat.



Quelles sont ces nouvelles interfaces graphiques ? Vous pouvez les afficher séparément en changeant la propriété app.maingui de votre projet.

app.maingui = gus.thread.gui.viewer

id	name	state	priority	group	daemon	deep	latest gus06 STE	first actual STE
14	AWT-EventQueue-0	RUNNABLE	6	main	false	38	gus06.entity.gus.thread.gui.v...	java.lang.Thread@getStackT...
15	TIMER_gus06.entity.gus.thre...	TIMED_WAITING	5	main	false	3	gus06.entity.gus.thread.gui.v...	java.lang.Object@wait (null:-2)
18	DestroyJavaVM	RUNNABLE	5	main	false	0	empty stack	empty stack

Threads number: 3

Il s'agit d'une entité de monitoring des threads de l'application

app.maingui = gus.system.prop.gui.viewer

System properties (55 values)	
key	value
awt.toolkit	sun.awt.windows.WToolkit
file.encoding	Cp1252
file.encoding.pkg	sun.io
file.separator	\
java.awt.graphicsenv	sun.awt.Win32GraphicsEnviron...
java.awt.printerjob	sun.awt.windows.WPrinterJob
java.class.path	C:\GUS\GUS06\TEST1\root\proj...
java.class.version	52.0
java.endorsed.dirs	C:\Program Files\Java\jre 1.8.0_...
java.ext.dirs	C:\Program Files\Java\jre 1.8.0_...
java.home	C:\Program Files\Java\jre 1.8.0_...
java.io.tmpdir	C:\Users\Augustin\AppData\Lo...
java.library.path	C:\Program Files\Java\jre 1.8.0_...

System env (46 values)	
key	value
==	::\
ALLUSERSPROFILE	C:\ProgramData
APPDATA	C:\Users\Augustin\AppData\Ro...
CLASSPATH	.;C:\PROGRA~2\UMF21~1.1E\li...
COMPUTERNAME	AUGUSTIN-VAIO
CYGWIN	tty
ComSpec	C:\Windows\system32\cmd.exe
CommonProgramFiles	C:\Program Files\Common Files
CommonProgramFiles(x86)	C:\Program Files (x86)\Commo...
CommonProgramW6432	C:\Program Files\Common Files
EMC_AUTOPLAY	C:\Program Files (x86)\Commo...
FP_NO_HOST_CHECK	NO
HOMEDRIVE	C:

Il s'agit là d'une entité qui affiche les propriété système de la JVM et ainsi que variables env liées à votre OS.

Pour information, ces deux entités se retrouvent dans l'espace "Supervision". Nous n'avons pas encore eu l'occasion de nous y intéresser mais cela viendra en temps voulu.



Les règles de mapping permettent en fait de fournir au gestionnaire des règles de génération de service dans des cas spécifiques d'appel d'entité.

La clé "tav.test.gridpanel22@tav.test.blackpanel" signifie par exemple :
appel réalisé par l'entité "tav.test.gridpanel22" avec l'identifiant "tav.test.blackpanel"

Nous pouvons généraliser en énonçant les 3 niveaux de règles qu'il est possible de déclarer dans le fichier de mapping, du plus spécifique au plus général :

Règle de niveau 1

<nom-entité>@<identifiant> = <règle-de-génération>

Un service demandé par l'entité <nom-entité> et identifié par <identifiant> doit être généré en appliquant la règle <règle-de-génération>

Règle de niveau 2

<pseudo>@<identifiant> = <règle-de-génération>

Un service demandé par une entité développée par <pseudo> et identifié par <identifiant> doit être généré en appliquant la règle <règle-de-génération>

Règle de niveau 3

@<identifiant> = <règle-de-génération>

Un service demandé par n'importe quelle entité et identifié par <identifiant> doit être généré en appliquant la règle <règle-de-génération>

Le mécanisme de résolution de mapping du gestionnaire est chargé de déterminer la règle de génération adéquate pour un appel particulier identifié par l'entité appelante et l'identifiant transmis. Comment procède-t-il ? Il recherche une règle de niveau 1 qui peut s'appliquer à l'appel, faute de quoi, il recherche une règle de niveau 2, puis une règle de niveau 3, et en dernier recours il considère que l'identifiant d'appel lui-même correspond à la règle de génération.

Et après ? Le gestionnaire tente d'interpréter la règle de génération ainsi obtenue, et de deux choses l'une :

- soit il est capable d'interpréter la règle et renvoie le service demandé
- soit il en est incapable et renvoie une exception

Dans le cas le plus simple, la règle de génération correspond au nom de l'entité souhaitée pour implémenter le service, mais il existe plein d'autres mécanismes de génération que nous aborderons plus tard.



17. Mise au point avant de poursuivre

A ce stade, une petite mise au point s'impose.

Je considère que vous avez acquis une autonomie suffisante sur le logiciel Gus Client pour qu'on puisse dorénavant se focaliser sur la programmation de nouvelles entités et la compréhension du gestionnaire GYEM.

Nous allons continuer à développer ensemble des entités de plus en plus complexes, qui nous permettront bientôt de concevoir de véritables applications.

Voici quelques bonnes pratiques souhaitables que vous devez avoir en tête pour avancer efficacement dans la suite de ce tutorial :

1. Naviguer entre les espaces des entités et des projets avec les raccourcis F11 / F12.
2. Faire évoluer le paramétrage du projet en éditant les fichiers prop et mapping.
3. Rechercher des entités et en créer de nouvelles au besoin.
4. Ne pas hésiter à dupliquer une entité existante (F3) pour reprendre du code existant
5. Exploiter toutes les fonctionnalités de l'éditeur d'entité pour coder plus rapidement (ctrl-e, ctrl-d, F2, et d'autres que vous verrons plus tard).
6. Faire ctrl-m, puis F9 pour valider régulièrement les modifications faites sur vos entités.

EN COURS DE REDACTION



II. Utilisation avancée

EN COURS DE REDACTION

